(2)

NEURAL NETWORKS AND THEIR POSSIBLE USE IN COMPUTER-ASSISTED DIAGNOSIS

J. DUNBAR
A. GINO

REPORT NO. 89-42

NAVAL HEALTH RESEARCH CENTER
P.O. BOX 85122
SAN DIEGO, CALIFORNIA 92186-5122

NAVAL MEDICAL RESEARCH AND DEVELOPMENT COMMAND
BETHESDA, MARYLAND

NEURAL NETWORKS AND THEIR POSSIBLE USE IN COMPUTER-ASSISTED DIAGNOSIS

Jean Dunbar, Ph.D.
Antonio Gino, Ph.D.

Medical Decision Support Department
Naval Health Research Center
P. O. Box 85122
San Diego, CA. 92138-9174

Report Number 89-42, supported by the American Society of Engineering Educators and by the Naval Medical Research and Development Command, Bethesda, Maryland, Department of the Navy, under research Work Unit M0095.005-6051. The opinions expressed in this paper are those of the authors and do not reflect the official policy or position of the Department of the Navy, Department of Defense, or the U. S. Government.

SUMMARY

Hospital corpsmen stationed on submarines frequently must make medical decisions without consulting a physician. Since evacuation decisions are extremely crucial and may involve aborting a mission, Computer-Assisted Medical Diagnosis (CAMD) is provided to confirm the need for medical evacuation. The Abdominal and Chest Pain modules deployed use a Bayesian approach to estimate the probability of a medical diagnosis. A variety of other algorithms, including various statistical tools and expert systems have been used elsewhere for this purpose. Currently a new tool, the neural network model, is being considered for CAMD. The purpose of this paper is to describe the mathematics involved in simulating a simple neural network and to discuss its potential in connection with the Navy's CAMD program.

NEURAL NETWORKS AND THEIR POSSIBLE USE IN COMPUTER-ASSISTED DIAGNOSIS

Introduction

Computer-Assisted Medical Diagnosis (CAMD) can be developed in a number of ways. Statistical approaches include Bayesian techniques, multiple regression, and discriminant functions. These reliable techniques have many advantages, but application and interpretation require some understanding of statistics[5] and assumptions concerning the distribution of the input data must frequently be made. Expert Systems offer an alternative which is easy to use, but time-consuming and expensive to set up and maintain.[6]

A new possibility in CAMD development is the neural network. Based on models of biological neural nets, a neural network can be defined as "...a computing system made up of a number of simple, highly interconnected processing elements, which processes information by its dynamic state response to external inputs".[1] In its true form, a neural network consists of hardware which uses parallel processing; (i.e., all the initial processing units execute simultaneously). However, neural networks can be simulated using sequential processing machines, where they have been studied for many years, especially in the fields of speech and pattern recognition. More recent technological advances and parallel processing capabilities have caused a renewed interest in neural nets and a flurry of new architectures (models of neural connections), some with capabilities not yet fully explored.[4]

This paper describes the mathematics involved in simulating a simple neural net. The possibility of using a neural net configuration for computer-assisted medical diagnosis is also discussed.

Definitions

A neural network is a data processing configuration which can be described by three basic parts: Architecture, Activation Function, and Learning Rule. The architecture is a description, or definition, of the interconnections of the elements of the system. The activation function

determines the way nodes "fire", and the learning rule describes the method used to determine parameters (weights) that will be used to compute the "output" from the network.

Architecture

The architecture consists of layers of processing units (or nodes) which have an inherent hierarchy. The input layer is composed of nodes which receive impulses from outside the network, while the output layer consists of nodes which send signals outside the network. Hidden layers, if present, receive and send signals only within the network. All layers are highly interconnected by directed arcs between nodes, while nodes in the same layer are usually not connected.

Suppose, for example, that input nodes 1 and 2 are connected directly to output node 3, with no hidden layer. Then weights w_{31} and w_{32} could be associated with arcs leading into node 3 from nodes 1 and 2 respectively. In general, the notation w_{ij} will be used for the weight of the arc into node i from node j. If the output from nodes 1 and 2 is o_1 and o_2, respectively, then the net input to node 3 will be the weighted sum of those outputs: $net_3 = w_{31}o_1 + w_{32}o_2$. The output from node 3 is produced by an activation function, f(x). So, the output from node 3 could be expressed as $o_3 = f(net_3)$.

Activation Function

The activation function may be as simple as the identity function, or it may be a more elaborate exponential function. A linear threshold activation function takes on values of either 0 or 1, depending on whether or not the input at that node has exceeded some pre-determined threshold. So, if T is a threshold value, and net_j is the total input to node j;

$$f(net_j) = \begin{cases} 0, & \text{if } net_j < T \\ 1, & \text{if } net_j \geq T \end{cases}$$

The biological analogy to this function is the neuron (node) that "fires" only when sufficiently stimulated.

Learning Rule

A collection of values for input nodes can be thought of as an input pattern. The pattern has been correctly "classified" if the value of the net's associated output nodes agrees with the expected or "target" output. However, before the network can correctly classify, it must be "trained".

Execution in a neural network may consist of two modes or phases. In the learning phase, input and expected (or target) output are presented to the network. The net processes the input to create calculated output which is compared to the target output. Based on the comparison, a learning rule is used to calculate new weights which will be used on the next iteration. Ideally, after a number of passes through the learning data the weights stabilize and the network is trained and ready for the second phase.

In the second phase (sometimes called recall mode), only input is fed into the network and weights are not updated. In recall mode, a net can "recognize a pattern" if it produces the expected output value(s) given a collection (or pattern) of input values. When a neural network is trained, the goal is for it to recognize all patterns from the training set and to generalize in a reasonable way by classifying similar patterns which were not part of training.

Networks trained by being furnished the correct output during the training phase are said to use "supervised learning". Networks are also differentiated on the basis of the type of input they are designed to process. Hopfield and Hamming nets, for example, are restricted to process binary input (see Lippman, 1987, for a review). However, greater flexibility is provided by neural nets which accept continuous-valued as well as binary input. The Navy's CAMD application would appear to benefit from using both discrete and continuous-valued inputs as well as supervised learning. Networks in this category include the Perceptron (which is a single layer neural network) and the Multi-Layer Perceptron, which uses the Generalized Delta Rule (see below).

Single Layer Neural Networks

The simplest neural nets have no hidden layers. The net is composed of N input nodes, M output nodes and no hidden layer. Every input node is connected to every output node, and no two nodes on the same layer are connected. The input nodes may be considered together as an N-dimensional

vector, so an input pattern would consist of an N-tuple of numbers. If we consider the weights leading into a given output node i as another N-dimensional vector, then net_i (the input to node i) is the dot product of the two vectors. That is, if o_j is the output from j nodes and w_{ij} is the weight of the arc from node j to node i, then the input to node i is the weighted sum of the outputs from all j nodes which are connected to it i.e.,

$$net_i = \Sigma_j w_{ij} o_j.$$

One learning rule which is error correcting, is called the Delta Rule. The change in weight between learning iterations is given by:

$$\hat{w}_{ij} = n (t_i - o_i) o_j, \text{ where;}$$

$\hat{w}_{ij}$ is the change in weight from node j to node i,
- n is a learning constant,
- o_i is the output from node i,
- o_j is the output from node j, and
- t_i is the target output from node i.

For input patterns which are linearly independent (no input pattern is a linear combination of the rest), this rule will provide learning for a neural net (i.e., it can perfectly learn any linearly independent training set).

When using a neural net with no hidden layers, the Delta Rule provides gradient descent and is equivalent to multiple linear regression from the input patterns to the targets. This can be verified mathematically by showing that if we define an error measure as;

$$E = 1/2 \Sigma_k (t_k - o_k)^2$$

then the derivative of this measure with respect to the weight vectors is negatively proportional to the change in weight $\hat{w}$ defined above.[4]

A geometric interpretation of the Delta Rule can be illustrated in the simplest case by visualizing error as a function of the weight vectors. In the case where the input has two nodes, for example, the sum of the mean squared error is a quadratic function of the weight vector. So, a graph of

mean squared error versus possible weight vectors is a parabola-shaped "bowl". The best weight vector, the one with least error, is one corresponding to the bottom of the bowl. The Delta Rule alters the current weight vector by adding a small amount to it in such a way that it's always moving down the slope of the paraboloid heading toward the bottom of the bowl.[2]

The Perceptron is an example of a neural network which has been studied in some depth. It has no hidden layers, its activation function is a linear threshold, and its learning rule is the Delta Rule, with the addition that o_i and o_j only take on values of either 0 or 1. Using a Perceptron, it has been proved that if a problem can be solved with no hidden units, then the Delta Rule will solve the problem.[4]

Many problems, however, cannot be solved without hidden units. The classic one is the "exclusive or" (XOR) problem which gives the output value 1 if exactly one of its two inputs is 1 as shown below:

inputs	output
00	0
01	1
10	1
11	0

This simple example cannot be solved in a single-layer network. That is, the network cannot be trained to recognize the four given input patterns. Intuitively, it's easy to see that a feature of the problem is that two very dissimilar input patterns, 00 and 11, are expected to have identical outputs. In general, neural nets with no hidden layers can only map similar input patterns to similar output patterns (i.e., they cannot map dissimilar input to identical output).[4]

Multiple Layer Neural Networks

One way to train a neural net to recognize the XOR rule is to use a hidden layer (Rumelhart et.al.[4] provide numerous other similar problems and solutions). Even though there does not yet exist one proven rule which can

be used to learn an arbitrary training set, the Generalized Delta Rule does provide a method which suffices for many configurations. In this rule, the change in weight from node i to node j is given by:

$$\hat{w}_{ji} = n\delta_j o_i \quad \text{where;}$$

n is the learning constant,

o_i is the output from node i, and

δ_j is an error factor

For output units $\delta_j = (t_j - o_j) f'(net_j)$, where;

t_j is the target at node j,

o_j is the output from node j,

net_j is the weighted sum at node j,

$f'(x)$ is the derivative of the activation function,

and for hidden units;

$$\delta_j = f'(net_j) \Sigma_k \delta_k w_{kj},$$

where a weighted sum of all errors above node j is taken.

Learning via the Generalized Delta Rule is accomplished in two phases. The first phase is a forward sweep where input values are presented to the input layer and are propagated forward through hidden layer(s) to the output units. Weighted sums are calculated and the activation function is used in this phase. Once the output nodes have been calculated, the second phase begins.

Since target information is available for the output layer, error calculation begins there when δ_j is calculated for each output node. Then the contribution to that error is calculated for each node in the previous hidden layer. The error is back propagated because δ_j is calculated recursively for each of the hidden layers. For each node i, after its δ_j is found, $\hat{w}_{ji}$ can be calculated and used for the next iteration. Thus, after one or more passes through a set of training data, the Generalized Delta Rule generates a set of weights which can be used to produce a mapping from input values to output values.[4] A net which uses the Generalized Delta Rule and back propagation is best used when learning can be done off-line in a non-real-time environment.[2]

In any gradient descent procedure, such as the Delta Rule, there is a hazard of oscillating near a local minimum. This problem is addressed in the Generalized Delta Rule by adding in a momentum term. One that is used frequently is a constant multiple of the previous $\hat{w}_{ij}$. This way, if the previous $\hat{w}_{ij}$ was negative, the current one is likely to be as well. So, in a sense, the momentum (or direction) is maintained. Another effect worthy of note is that of the learning constant, n. The value of this constant should lie between 0 and 1, with smaller values causing learning to proceed very slowly and larger values perhaps contributing to an oscillating effect.

The activation function appropriate for use with a Generalized Delta Rule should be continuous, increasing, and non-linear (sometimes called a sigmoidal function). One that works well is given by:

$$f(x) = 1 \; / \; (\; 1 + e^{-x} \;).$$

It's easy to show that this function has the derivative

$$f'(s) = f(s)[\; 1 - f(s)].$$

This makes computations somewhat simpler because, based on standard net definitions, $f(net_j) = o_j$. So, in the Generalized Delta Rule formulas above:

$$f'(net_j) = f(net_j)[\; 1 - f(net_j)] = o_j \; [\; 1 - o_j].$$

A threshold is sometimes incorporated into the activation function by modifying it slightly as:

$$f'(x) = 1 \; / \; (\; 1 + e^{-x \; + \; T} \;).$$

The effect is to suppress noise. That is, if the value of the weighted sum is not as large as T , then the value of f(x) is effectively diminished. One way to accomplish this is to add a bias node whose value is always 1 and whose weight is modified just like other nodes. This way the value for T is calculated along with other weights.

Examples

Example 1. Single Layer Neural Net

In the sample shown in Appendix I, it may be noted that after stepping through 4 iterations feeding in the training input and expected target output for the XOR problem, the weights are modified according to the formulas shown. After 2000 iterations of training on the XOR data, the weights have been further modified, but when executing the neural net in recall mode, it is clear that the expected outputs have not been learned.

Example 2. Multiple Layer Neural Net

In Example 2 (Appendix II), a hidden layer with one node has been inserted between the input layer and the output layer. The hidden layer allows the neural net to develop an internal representation for its patterns. The output from node 4 is computed as in the single layer net above, and this value is passed into node 5 along with input from nodes 2, 3, and the bias node. At this point the error at node 5 is calculated and this value is back propagated to the hidden layer. After the error at a layer is calculated, new values for its associated weights can be obtained for any subsequent iterations. The significance of this example is that the XOR pattern is learned after passing through the training data 2000 times and that the outputs indicate the network has learned the pattern well.

Discussion

A basic strategy for providing a neural network for CAMD support for corpsmen might include using a body of case history information to develop relevant weights and parameters. Analysis could be done on a central mini or mainframe computer with the results down-loaded to laptop computers and updated on a periodic basis.

The case history information could be used to train a neural network to recognize "patterns" of symptoms and classify them into diseases. Then, the stabilized network with the resulting weights could be validated using an equivalent (but not identical) collection of case histories and comparing the network's classifications to the actual diagnoses.

A feasible architecture would include an input node for each symptom and an output node for each disease. A hidden layer would be used to provide internal representation. Part of the design and training phase would include deciding on the number of nodes in the hidden layer and the size of any thresholds and learning constants used. The Generalized Delta Rule used with the back propagation of errors and with the exponential activation function described above, would provide for supervised learning with continuous-valued input.

Disadvantages of using this method include collecting the body of case history information and designing the neural net itself. We know that for linearly independent sets of input patterns, the Delta Rule and back propagation can perfectly learn any training set. This means that whenever the patterns are fairly distinct, learning is automatic and guaranteed. Otherwise, the Delta Rule will give a best possible fit, but not an exact one.[4] Since real-world situations are rarely linearly independent, designing the architecture of the network itself is not a well-defined process with a straightforward algorithm.

In comparing neural nets to statistical methods, Lippman focuses on the learning potential of the former: "Traditional statistical techniques are not adaptive but typically process all training data simultaneously before being used with new data. Neural net classifiers... also make weaker assumptions concerning the shapes of underlying distributions than traditional statistical classifiers. They may thus prove to be more robust when distributions are generated by nonlinear processes".[3] If neural nets using the Delta Rule are developed to be used on laptops aboard ships, all training data would be processed ahead of time, so this capability of neural nets would not be used. However, being freed from making any assumptions about the underlying distributions and being able to handle continuous-valued input are two features which could be used in this application. Also, case history data could be used with a minimal amount of reformatting and data analysis.

A rigorous statistical comparison of the accuracy of a neural network with other systems will no doubt help to detect any shortcomings and explore the usefulness of such systems as a tool for Computer-Assisted Medical Diagnosis.

References

1. Caudill, M., "Neural Networks Primer Part I", AI Expert, 46-52, December 1987.

2. Caudill, M., "Neural Networks Primer Part III", AI Expert, 53-59, June 1988.

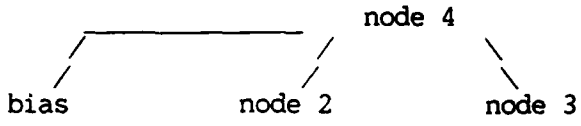
3. Lippmann, R.P., "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, 4-22, April 1987.

4. Rumelhart, D.E., McClelland, J.L., and the PDP Research Group, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MIT Press, 1986.

5. Sicoly, F., "Computer-Aided Decisions in Human Services: Expert Systems and Multivariate Models", Computers in Human Behavior, Vol 5, 47-60, 1989.

6. Spiegelhalter, D.J. and Knill-Jones, R.P., "Statistical and Knowledge-based Approaches to Clinical Decision-support Systems, with an Application in Gastroenterology", J. R. Statist. Soc. A, 147, Part 1, 35-77, 1984.

Appendix I



node 2			node 3			Bias			node 4		
o	w	$\hat{w}$	o	w	$\hat{w}$	o	w	$\hat{w}$	sum	o	err
	0	1	.0	.0	1	.0	1	1	.0		
1	.9613	-.0387	1	.9613	-.0387	1	.9613	-.0387	3	.953	-.043
1	.9508	-.0105	0	.9380	-.0232	1	.9508	-.0105	1.923	.872	.014
0	.9445	-.0063	0	.9241	-.0139	1	.8140	-.1368	.951	.721	-.145
0	.9408	-.0038	1	.9329	.0088	1	.7491	-.0649	1.738	.850	.019
					after 2000 iterations of XOR training data						
1	-.0713	-.0878	1	-.1065	-.0549	1	.1322	-.0351	.132	.533	-.133

input	output
00	.533
01	.506
10	.515
11	.489

NOTE: When calculating a weighted sum, the previous weights are used

sum at node 4:

$$sum_4(n+1) = o_2(n+1)\, w_{42}(n) + o_3(n+1) w_{43}(n) + b\, w_{4b}(n)$$

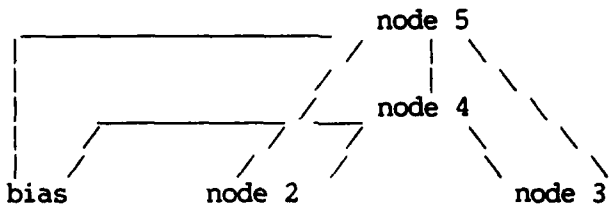
output from node 4: $o_4 = 1 / (1 + e^{-sum_4})$

error at node 4: $err_4 = [t_4 - o_4]\, o_4\, [1 - o_4]$

weight change: $\hat{w}_{4i}(n+1) = .9\, err_4\, o_i + .6 \hat{w}_{4i}(n)$

new weights : $w_{ij}(n+1) = w_{ij}(n) + \hat{w}_{ij}(n+1)$

Appendix II



node 2			node 3			Bias			node 4		
o	w	ŵ	o	w	ŵ	o	w	ŵ	sum	o	err
0	1	.0	0	1	.0	1	1	.0			
1	.9993	-.0007	1	.9993	-.0007	1	.9993	-.0007	3	.953	-.001
1	.9991	-.0002	0	.9989	-.0004	1	.9991	-.0002	1.999	.881	.0003
0	.9990	-.0001	0	.9986	-.0003	1	.9798	-.0193	.999	.731	-.021
0	.9989	-.0001	1	.9988	.0002	1	.9686	-.0112	1.978	.879	.0003

after 2000 iterations of XOR training data

1	6.9605	.0003	1	6.9669	.0005	1	-2.882	.0003	11.045	1	0

node 2			node 3			node 4			bias			node 5		
o	w	ŵ	o	w	ŵ	o	w	ŵ	o	w	ŵ	sum	o	err
0	1	.0	0	1	.0	1	1	.0	1	1	0			
1	.9837	-.0163	1	.9837	-.0163	.95	.9844	-.0156	1	.9837	-.0163	3.95	.981	-.018
1	.9765	-.0072	0	.9739	-.0098	.88	.9774	-.007	1	.9765	-.0072	2.83	.945	.003
0	.9722	-.0043	0	.9680	-.0059	.73	.9002	-.0772	1	.8723	-.1042	1.69	.844	-.111
0	.9696	-.0587	1	.9683	.0003	.88	.8572	-.043	1	.8136	-.0587	2.63	.933	.004

after 2000 iterations of XOR training data

1	-5.3487	-.004	1	-5.3539	-.003	1	11.57	-.0016	1	-3.3716	-.0023	-2.5	.076	-.005

input	output
00	.060
01	.934
10	.934
11	.076

Back propagation

error at node 5 (or any output node)

$$err_5 = [t_5 - o_5]\, f'(sum_5) = [t_5 - o_5]\, o_5\, [1 - o_5]$$

error at node 4

$$err_4 = (w_{54})(err_5)\, f'(sum_4) = (w_{54})(err_5)\, o_4\, [1 - o_4]$$

error at node j in any hidden layer

$$err_j = \sum_i (w_{ij})(err_i)\, f'(sum_j)$$

i summing over all nodes i above node j

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NHRC Report No. 89-42	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Health Center 6b OFFICE SYMBOL (*If applicable*) Code 20	7a. NAME OF MONITORING ORGANIZATION Commander Bureau of Medicine and Surgery
6c. ADDRESS (*City, State, and ZIP Code*) P.O. Box 85122 San Diego, CA 92138-9174	7b. ADDRESS (*City, State, and ZIP Code*) Department of the Navy Washington, DC 20372-5120
8a. NAME OF FUNDING / SPONSORING ORGANIZATION National Naval Medical Center 8b OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*) Capital Region Bethesda, MD 20814	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 63706N; PROJECT NO. M0095; TASK NO 005; WORK UNIT ACCESSION NO. 6051
11. TITLE (*Include Security Classification*) (U) NEURAL NETWORKS AND THEIR POSSIBLE USE IN COMPUTER-ASSISTED DIAGNOSIS	
12. PERSONAL AUTHOR(S) Dunbar, J. Ph.D., Gino, A., Ph.D.	
13a. TYPE OF REPORT Final 13b TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (*Year, Month, Day*) 15 PAGE COUNT
16. SUPPLEMENTARY NOTATION	
17 COSATI CODES FIELD / GROUP / SUB-GROUP	18 SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*) Computer-Assisted Medical Diagnosis, Neural Network Models, Error Back propagation, Computations

19 ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Hospital corpsmen stationed on submarines frequently must make medical decisions without consulting a physician. Since evacuation decisions are extremely crucial and may involve aborting a mission, Computer-Assisted Medical Diagnosis (CAMD) is provided to confirm the need for medical evacuation. The Abdominal and Chest Pain modules deployed use a Bayesian approach to estimate the probability of a medical diagnosis. A variety of other algorithms, including various statistical tools and Expert Systems have been used elsewhere for this purpose. Currently a new tool, the neural network model, is being considered for CAMD. The purpose of this paper is to describe the mathematics involved in simulating a simple neural network and to discuss its potential in connection with the Navy's CAMD program.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Antonio Gino, Ph.D.	22b TELEPHONE (*Include Area Code*) (619) 553-8393 22c. OFFICE SYMBOL Code 20

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE